Description of Data Sets and Generators

Version 1.0

Frédéric Quesnel
Atoosa Kasirzadeh
François Soumis

June 2014

Contents

1 Terminology

Here is a brief description of different terms related to the instances:

- Base : A base is an airport where crew members are stationed. Crew members start and end their pairing at their respective base.
- Airleg : An airleg is a direct flight between two airports.
- Deadhead : A deadhead is a flight that a crew member takes only for relocation. The crew member does not perform any work during this flight.
- Duty : A duty is a day of work for a crew member, composed of one or multiple airlegs (and/or deadheads). A duty ends when the crew member is granted a sleep rest (layover).
- Pairing : A sequence of duties and layovers for an unspecified crew member that starts and ends at the same base.
- Credit : The credit is a measure of the time worked by a crew member. The amount of credit for a pairing is the duration of scheduled flight, plus half the duration of deadheads (if any).

2 Description of Data Sets

2.1 Note on the Data

All the data we present here comes from a major North-American airline, except for the data generated by the generators described below. For confidentiality reasons, all airport names and flight numbers have been changed. Furthermore, months and years are all set to January 2000.

The data in itself was not altered. Only the carry-in and carry-out were deleted, since those flights can not be possibly covered.

2.2 Airports and Bases (listOfBases.csv)

This file contains information about the airports used in the instances. Each line (except for first and last) gives the name , status, and number of employees associated to each airport. If the status is set to one, it means that the airport is a base. If it is set to zero , the airport is not a base.

It is structured as follows:

```
airport , status , nbEmployees
[Acronym_Of_Air_1] , [Status_Of_Air_1] , [Number_Of_Employees_In_Air_1]
[Acronym_Of_Air_2] , [Status_Of_Air_2] , [Number_Of_Employees_In_Air_2]
```

```
[Acronym_Of_Air_3] , [Status_Of_Air_3] , [Number_Of_Employees_In_Air_3]
etc.
```

Where :

- `[Acronym_Of_Airport_x]` : acronym representing the airport x.
- `[Status_Of_Airport_x]}` : 0 if the airport is not a base and 1 if it is.
- `[Number_Of_Employees_In_Airport_x]`:integer corresponding to the number of employees in base x.

2.3 Airlegs (day_x.csv)

There is one file for scheduled flights for each day of the horizon. It is structured as follows:

```
leg_nb , airport_dep , date_dep , hour_dep , airport_arr , date_arr , hour_arr
LEG_[x]_[nb] , [dep_air] , [dep_date] , [dep_time] , [arr_air] , [arr_date] , [arr_time]
LEG_[x]_[nb] , [dep_air] , [dep_date] , [dep_time] , [arr_air] , [arr_date] , [arr_time]
LEG_[x]_[nb] , [dep_air] , [dep_date] , [dep_time] , [arr_air] , [arr_date] , [arr_time]
etc.
```

Where:

- `[x]` : day of the horizon, in 2-digits format (e.g. 02).
- `[nb]` :flight number for current day (note that numbers are not necessarly successive).
- `[dep_air]` :acronym representing the departure airport.
- `[dep_date]` : departure date in YYYY-MM-DD format.
- `[dep_time]` : departure time in hh:mm format.
- `[arr_air]` : acronym representing the arrival airport.
- `[arr_date]` : arrival date in YYYY-MM-DD format.
- `[arr_time]` : arrival time in hh:mm format.

2.4 Maximum Credit constraints (credit_constrains.csv)

This file contains the total number of credits for an initial solution and gives the maximum credit allowed for each base, for different values of the percentage of the credit allowed to the main base. The file is in csv format. The structure of the file is explained in the section 3.3.2.

2.5 Crew Availability per Base per Day (crew_avail_const.csv)

The files are tables containing the number of crew members available for each base and each day. The total crew availability is taken from a previous solution of the scheduling problem. The total number of crews available is distributed among the bases. The structure of the file is explained in the section 3.4.3.

3 Generators

This section describes the generators. The generators are provided to allow researchers to create custom data, with different parameters (% of slack to add, for example). The data included was generated using the parameter values listed in the file "params.txt".

3.1 Preferred Vacations Generator

This generator assigns random vacation dates for fictive employees. The number of employees with vacation in the horizon corresponds to a certain percentage of all employees. Their preferred vacation periods can occur between 2000/01/01 and 2000/01/31, with a length between 2 and 15 days, with uniform distribution. The vacation periods start at 00:00 and end at 23:59. The output file is in csv format.

3.1.1 Generator

The program generating the preferred vacations is the file "preferredVacations.cpp". It creates the preferred vacation files for multiple instances at once. The file "listOfBases.csv" is used to find the name of the bases. It is also possible to change the percentage of employees with vacations in "params.txt".
Files that are required in the instance directory:

- listOfBases.csv

Parameter to change in params.txt (inside brackets):

```
instances
[path/to/instance]
[path/to/another/instance]
[...]

...

preferredVacations.cpp
% employees choosen : [value]
```

where :

- `[path/to/instance]`:path to an instance folder. There can be an unlimited number of path, each on a different line. The list of folders for the instances must be followed by an empty line (otherwise, subsequent lines will be treated as an instance folder as well).

- `[value]` : The percentage of employees to whom vacations are assigned. It must be a number between 0 and 100.

compilation in linux terminal:

```
    g++ preferredVacations.cpp -o preferredVacations
```

execution in linux terminal :

```
    ./preferredVacations path/to/params.txt
```

3.1.2 Output File

The program creates the file named personalizedEmployees.csv. Each line provides information on the employee base, the start and the end date of his/her vacation. Each vacation starts at 00:00 and ends at 23:59.
The file is structured as follows:

```
employee , base , vacationName , startDate , startHour , endDate , endHour
EMP001 , [BASE_NAME] , Vacation_001 , [Start_Date] , 00:00 , [End_Date] , 23:59
EMP002 , [BASE_NAME] , Vacation_002 , [Start_Date] , 00:00 , [End_Date] , 23:59
EMP003 , [BASE_NAME] , Vacation_003 , [Start_Date] , 00:00 , [End_Date] , 23:59
etc.
```

Where :

- `[BASE_NAME]` : acronym representing the employee's base.
- `[Start_Date]` : Starting date of the vacation, in YYYY-MM-DD format.
- `[End_Date]` : end date of the vacation, in YYYY-MM-DD format.

3.2 Preferred Airlegs

This set of constraints randomly assigns preferred airlegs to fictive employees. Each employee prefers a certain percentage of all airlegs associated to his base. An airleg f is related to a base b if, in a given set of pairings, a pairing starting at b contains f. The main reason why the preferences are selected in this way (instead of randomly selecting airlegs from a list of all airlegs, for example) is to obtain realistic data. In fact, it is reasonable to assume that, most of the time, crew members will prefer legs close to their home base, and usually, pairings linked to a base contain airlegs involvings airports near that base.

3.2.1 Generator

The program generating the preferred vacations is the file "EmployeeLegPreferences.cpp". It creates the crew preferences for multiple instances at once. The program uses the file "listOfBases.csv" to obtain the number of employees assigned to each base. The file "initialSolution.in" contains all the pairings obtained in a previous solution of the pairing problem. It is used to link the airlegs to their respective bases.
Files that are required in the instance directory:

- listOfBases.csv
- initialSolution.in

Parameters to change in params.txt (inside brackets):

```
instances
[path/to/instance]
[path/to/another/instance]
...

EmployeeLegPreferences.cpp
% airleg choosen : [value]
```

where :

- `[path/to/instance]`: path to an instance folder. There can be an unlimited number of paths, each on a different line. The list of folders for the instances must be followed by an empty line (otherwise, subsequent lines will be treated as an instance folder as well).
- `[value]` : The percentage of airlegs chosen for each employee. It must be a number between 0 and 100.

compilation in linux terminal:

```
g++ EmployeeLegPreferences.cpp -o EmployeeLegPreferences
```

execution in linux terminal :

```
./EmployeeLegPreferences path/to/params.txt
```

3.2.2 Output File

The program creates the file named PreferredAirLegs.csv. Each line is a list of the preferred airlegs for one employee.

The file is structured as follows:

```
employee , legs
EMP001   , LEG_[day]_[number] , LEG_[day]_[number] , ...  , LEG_[day]_[number]
EMP002   , LEG_[day]_[number] , LEG_[day]_[number] , ...  , LEG_[day]_[number]
EMP003   , LEG_[day]_[number] , LEG_[day]_[number] , ...  , LEG_[day]_[number]
etc.
```

Where :

- `[day]` : day of the period where the flight is flown.
- `[number]`: flight number (unique for that day ; see section "Airlegs")

3.3 Maximum Credit per Base

This set of constraints limit the credit (number of flight hours) for each base. We consider the case where most part of the credit goes to one "main" base, while the rest is distributed evenly among the other bases. The calculation of the total credit allowed and the determination of the main base is done by looking at previous results with the same instance, with no constraint on the credit. Note that a small slack is added to the total credit in order to facilitate feasability.

3.3.1 Generator

This generator provides the credit constraints for multiple instances. The file "creditedHours" is used to compute an estimation of the credit required for each base. The file "listOfBases" is also used to confirm the name of the bases. Since the values of the credit listed in the file "creditedHours" also take into account the briefing/debriefing time, 2 hours were removed from the total credit for each duty. The number of duties was computed using the file solution_0. Finally, it is possible to change the slack added to the total and the percentage of credit to main base in "param.txt".

Files that are required in the instance directory:

- creditedHours
- solution_0
- listOfBases.csv

Parameters to change in params.txt (inside brackets) :

```
instances
[path/to/instance]
[path/to/another/instance]
...

credit_constrains.cpp
% slack : [slack]
% to main base : [value]
```

where :

- `[path/to/instance]`: path to an instance folder. There can be an unlimited number of paths, each on a different line. The list of folders for instances must be followed by an empty line (otherwise, subsequent lines will be treated as an instance folder as well).
- `[slack]` : number representing the amount of slack to be added to the total credit. For example, if there is a total of 1000 credited hours, and percentSlack = 3 , 30 hours will be added, for a total of 1030 hours to be distributed among the bases.

- `[value]` : percentage of the credit that will be given to the main base. "-1" gives the same distribution as in the file solution_0.

compilation in linux terminal:

```
    g++ credit_constrains.cpp -o credit_constrains
```

execution in linux terminal :

```
    ./credit_constraints path/to/params.txt
```

3.3.2 Output File

The output of this program is the file named "credit_constrains.csv". It describes a possible distribution of the credit among the bases. The file is structured as follows:

```
"slack added : [slack]%"

base                , [BASE1]        , [BASE2]        , [BASE3]      , ETC
["5%/90%/5%"]       , [CREDIT1]      , [CREDIT2]      , [CREDIT3]    , ETC
```

Where :

- `[slack] \verb` : slack added to the values.
- `[BASEx]` : acronym representing the base x.
- `[CREDITx]` : maximum credit allowed to base x, given the ratio of the first column.
- `["5\%/90\%/5\%"]` :percentage of credit allowed for each base. (e.g. 5%/90%/5% : 5% to base 1, 90% to base 2, 5% to base 3). The last row represents an even distribution among all bases. The first row represents the initial distribution.

3.4 Crew Availability per Base per Day

This set of constraints limits the crew availability for each base and each day. The total number of crews is based on a previous solution. The constraints limit the number of duties taking place each day and for each base. In one case, the values of the previous solution are directly displayed (with added slack). We also propose a version where the crew availabilty is constant troughout the month for each base (also with slack added to the total crew availability). In this scenario, the values from the previous solution are averaged per base.

3.4.1 Averaging and Rounding

When averaging the crew availability for each base over the whole period, we want to have integer values for each base, while making sure the total availability stays unchanged.

In order to do so, we first calculate the average crew availability for each base. The bases that have averages with the higest decimal values are then rounded up, and the others are rounded down. In order to determine the number of bases to round up, the following calculation is made :

$$N = \left\lceil \sum_b A_b \right\rceil - \sum_b \lfloor A_b \rfloor$$

where N is the numer of bases to round up and A_b is the average crew availability for base b.

For example, in the following situation :

Base	average
BASE1	39.4
BASE2	21.3
BASE3	12.7

$$\sum_b A_b = 73.4$$

$$\sum_b \lfloor A_b \rfloor = 39 + 21 + 12 = 72$$

We must then round up $74 - 72 = 2$ bases
BASE1 and BASE3 are rounded up since they have the highest decimal values.
The final values are:

Base	average
BASE1	40
BASE2	21
BASE3	13

It can be shown that the number of bases to round up is always smaller than the number of bases. The total crew availability will always be within 1 crew/day of the original values.

A similar procedure is applied in the case where the distribution is unchanged. The procedure is applied for each day separately. In the case where more than one base have the same availability, and one of these bases must be rounded up, the base is chosen randomly to avoid any bias in favor of one base.

Exception : In the rare case where all averages are integers, one base has its average rounded up anyways.

3.4.2 Generator

One file is created by the generator: crew_availabil_const.csv. The total crew availability is determined by counting the number of duties for all the crew members, for each base and for each day in a previous solution to the scheduling problem. This information is available in the file "solution_0". The program uses the file "listOfBases.csv" to extract the names of the bases. The files "day_x.csv" are used to determine the number of days in the period.
Files that are required in the instance directory:

- listOfBases.csv

- day_x (multiple files)
- solution_0

Parameters to change in params.txt (inside brackets):

```
instances
[path/to/instance]
[path/to/another/instance]
[...]

crew_availability_constrains.cpp
% slack : [slack]
% main_base : [percent]
```

where :

- `[path/to/instance]`:path to an instance folder. There can be an unlimited number of paths, each on a different line. The list of folders for instances must be followed by an empty line (otherwise, subsequent lines will be treated as an instance folder as well).
- `[slack]`:percentage of slack to add to the total crew availability.
- `[percent]`:percentage of the crew availability to be added to the main base. If set to "-1", the crew availability will be the same as in the previous solution, with slack added.

compilation in linux terminal:

```
g++ crew_availability_constrains.cpp -o crew_availability_constrains
```

execution in linux terminal :

```
./crew_availability_constraints path/to/params.txt
```

3.4.3 Output File

The output file produced (crew_availabil_const.csv) contains the crew availability per base.
The file isd as below:

```
"Number of crews available at each base for each day of the period. [slack]% slack added."

base    , [BASE1]    , [BASE2]    , [BASE3]
Day1    , [#1_1]     , [#1_2]     , [#1_3]
Day2    , [#2_1]     , [#2_2]     , [#2_3]
etc.
Day31   , [#31_1]    , [#31_2]    , [#31_3]
```

where:

- `[slack]`:slack added to the values.
- `[BASEx]`:acronym representing the airport x.
- `[#i_j]`: number of crew members available on day i at base j.

4 Description of the Files Used by the Generators

4.1 params.txt

This file contains the parameters necessary to run the generators :

- preferredVacation
- EmployeeLegPreferences
- credit_constrains
- crew_availability_constrains

In this file, the user may specify the instance folder where he/she wishes to run one program. It is also possible to modify the parameters used to run each program.

4.2 creditedHours

This file contains the information about the schedule of employees, from a previous solution of the scheduling problem (without credit and crew availability constrains). It is used to approximate the number of credited hours required for each base.The file contains information about the base, the number of credited hours, the schedule cost and the number of vacation days for each employee.

4.3 initialSolution.in

This file correspond to the initial solution of the pairing problem. Each line (except for first and last) contains the base and the airlegs corresponding to one pairing. The file is used to link each airleg to a base when generating airleg preferences for the employees.

4.4 listOfBases.csv

This file contains information about the airports used in the instances.

Each line (except for first and last) gives the name , status, and number of employees associated to one airport. The status determines if the airport is a base or not.

For more details on the structure of the file, see the section 2.2.

4.5 solution_0

This file contains the detailed schedules for all crew members from a previous solution of the scheduling problem.

It is used to approximate the number of duties required for each base.

4.6 day_x.csv

This file contains information on the airleg to be flown on day x.

For more details on the structure of the file, see the section 2.3.